阿里云开发者社区
ALIBABA CLOUD DEVELOPER COMMUNITY

Dubbo 分布式服务治理实战

阿里云开发者学院推荐配套教材

作者：侠客

- Dubbo分布式架构原理
- 模拟淘宝双11大规模服务集群治理
- Dubbo3.0高性能微服务的发展方向

藏经阁

阿里云开发者电子书系列

扫一扫

免费领取同步课程

钉钉扫一扫

进入官方答疑群

开发者学院【Alibaba Java 技术图谱】

更多好课免费学

阿里云开发者“藏经阁”

海量电子书免费下载

推荐理由

Dubbo 是阿里巴巴开源的高性能分布式 RPC 服务治理框架，已经捐献给 Apache 开源组织。最新的版本是 3.0。在阿里巴巴、工行、电信、银联、中国人寿、网易、滴滴当当等互联网公司中有大规模使用，历经双 11 等流量检验。Dubbo 提供了六大核心能力：面向接口代理的高性能 RPC 调用，智能容错和负载均衡，服务自动注册和发现，高度可扩展能力，运行期流量调度，可视化的服务治理与运维，功能强大。未来支持 Go、K8S 云原生等技术。代表着高性能微服务架构的发展方向。

目录

1. Dubbo 分布式框架介绍与 3.0 新特性

内容简介：

一、Dubbo 课程环境介绍

二、Dubbo3.0 分布式服务治理框架新特性

一、Dubbo 课程环境介绍

1. 阿里 Java 开发者学院 2021 最新课程

Dubbo 相比传统分布式框架有很大差别，Dubbo 本身解决的问题不仅仅是服务调用，致力于提供高性能和透明化的 RPC 远程服务调用方案，以及 SOA 服务治理方案。所以相比传统 RPC，在架构层次上又做了一次很重要的升级，这个课程是作为 Java 服务课程前置的非常重要的学习课程。

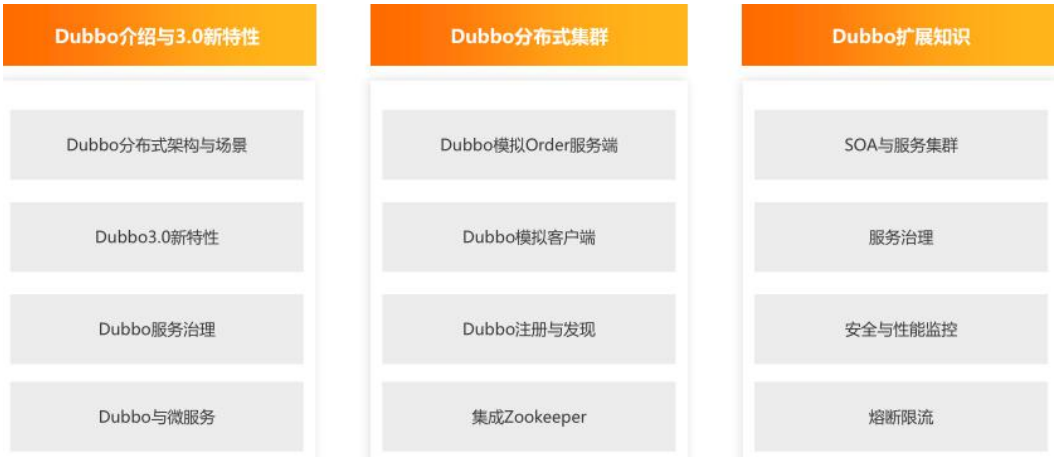
阿里巴巴专家亲自授课，覆盖最新 Spring Cloud 微服务架构：

2. Dubbo 分布式服务治理框架实战—步步为赢

如果是刚工作不久的同学，可以看一下 Java 的高级编程知识，包括我虚拟机、多线程等编程框架。希望大家在做架构师对公司框架有深入、系统的研究。回顾整个 Dubbo 分布式框架，在开源社区的影响非常大，主要在微服务架构流行之前，实际已经在做大规模的分布式服务集群的开发、落地、治理工作。Dubbo 很多经典设计模式和思想挑战性问题，目前在微服务架构中用到了相同的技术条件和问题，有很重要的参考、对比价值。

在做架构师技术选型的时候，不能只会一个框架，很多公司的框架实际是定死的，作为架构师照搬就行。但是对于架构师的要求来说，还是希望能够精通多种架构方式，这样在应对不同的问题、不同的场景的时候，能有更多合理性选择。

Dubbo 从理论到架构设计模式，逐步带入实战练习，模拟订单服务、模拟机群，客户端实现调用操作。在实战过程中，希望大家把其他的组件给用起来，比如说 Dubbo 注册中心等，逐步联系起来。当然 Dubbo 本身也有自己的监控组件，有自己的熔断机制。



二、Dubbo3.0 分布式服务治理框架新特性

1. 分布式服务治理框架 Dubbo 介绍

Dubbo 已经发展到 3.0，经过很长时间迭代，在阿里巴巴集团内部大量使用，在官网上有介绍。回顾整个分布式架构发展历史，淘宝诞生在 2003 年，经历过非常典型的各种分布式架构发展过程，淘宝本身有自己的分布式框架 hsf，hsf 和 Dubbo 设计思想有很多相似之处，协议、集群治理也有很多重合的地方。但是 Dubbo 本身是使用的更广泛，在业界影响力更大。

Dubbo 优秀的地方在于，整个架构诞生在 SOS 时代，但是有 RPC 思想的影子,相比传统的 RPC 框架，Dubbo 走的路线更高了一个层级，做大规模服务的解耦和治理的工作，管控服务，实现服务的查找、发现、负载均衡、上线下线以及流量管控等功能。

今天来看，不仅阿里巴巴，行业里面有很多公司在用 Dubbo 框架，阿里在 2017 年又重新启动了 Dubbo 开源工作，发展的越来越完善。但是 Dubbo 本身并不是一个强大的体系，还要借助其他的技术组件，实现整个服务治理工作。

分布式服务治理框架 Dubbo 介绍总结：

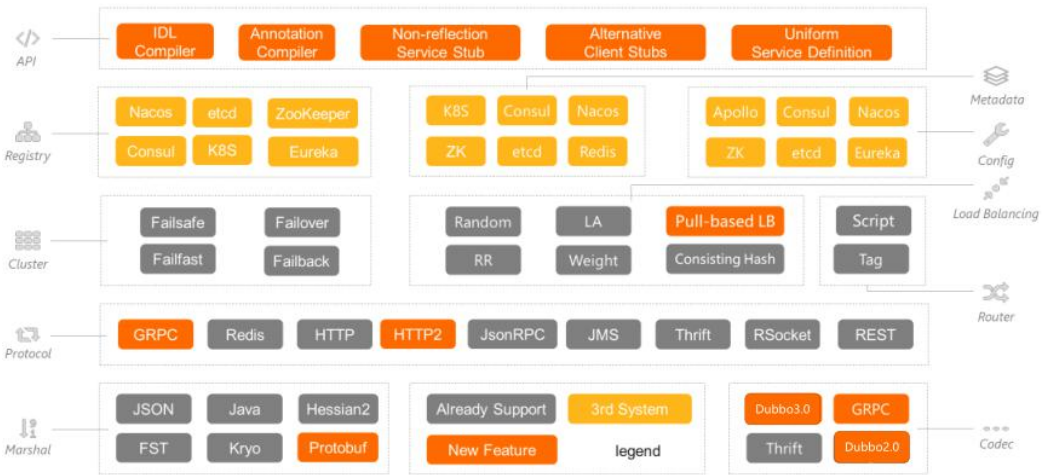
1）Dubbo 是一个阿里巴巴开源的分布式 RPC 服务治理框架；
2）致力于提供高性能和透明化的 RPC 远程服务调用方案；
3）是阿里巴巴 SOA 服务化治理方案的核心框架；
4）每天为 2,000+个服务提供 3,000,000,000+次访问量支持；
5）并被广泛应用于阿里巴巴集团的各成员站点。
6）官方网站 http://dubbo.io/；
7）源码：https://github.com/alibaba/dubbo；
8）下载：http://repo1.maven.org/maven2/com/alibaba/dubbo；
9）微博：http://weibo.com/dubbo。

2. Dubbo 发展历程

Dubbo 2008 年诞生，后面有一段时间不维护了，开源项目也会受到成员工作繁忙程度等因素的影响。2017 年又重新启动，进入孵化器。2020 年启动 3.0，开启微服务时代，对 Dubbo 进行升级扩充，在微服务时代 Dubbo 向新的层级进行兼容和支持。

3. Dubbo3.0 —— 架构升级扩展:多协议+云原生

Dubbo 之前也支持各种不同的通讯协议、虚拟化协议，而且部署方式相比的 HSF 更多元化、更灵活。目前在设计角度很明显的增强，是开始支持 GRPC 行业级协议，成为行业标准。很多接口都用包括 K8S，大量调度。另外像 HTTP2，是 HTTP 协议新的升级版本，包括负载均衡策略以及路由策略都定了增强，这些 Dubbo 做的比较好的地方。Dubbo 在 3.0 以后，整个生态会变得更强大，多协议向云原生靠拢。



4. Dubbo 服务治理框架

整体来说，Dubbo 本身是作为一个非常优秀的服务治理，服务治理指的并不是简单 SOA 架构，指的是大规模服务集群的量级，是 Dubbo 较强大的地方。当然如果只有一个服务两个服务的话，Dubbo 也可以做，理论上可以支持更高、更复杂的架构。Dubbo 服务治理框架总结：

1）SOA 架构解决架构的异构交互问题；

2）可以实现不同语言直接功能接口的重用；

3）普通的服务框架解决开发 Web 服务接口开发；

4）服务注册+ 服务管理+ 服务发现+健康监控；

5）Dubbo 是一个分布式 RPC SOA 服务治理框架；

6）致力于提供高性能和透明化的 RPC 远程服务调用方案；

7）是阿里巴巴 SOA 服务化治理方案的核心框架；

8）每天为 2,000+个服务；

9）提供 3,000,000,000+次访问量支持；

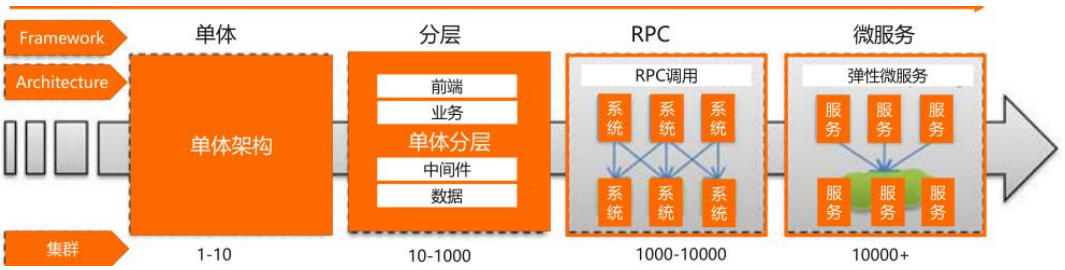
10）并被广泛应用于阿里巴巴集团的各成员站点。

5. 阿里巴巴 Dubbo 架构演化

Dubbo 诞生的时间比较早，做架构设计的时候，看到包括单体、分层、分布式架构、微服务架构等，Dubbo 是在不断的做迭代、扩展、演化。在架构改造上 Dubbo 是越来越丰富，协议上是越来越越复杂，也支持多元化。

Java Spring Cloud 目前来看，主要还是 REST API 协议，对于公网是一个很好的选择，但是对于局域网，尤其是系统架构全部是 Java，统一语言的场景的情况下，用同一种协议更好。比如更接近于 TPP 协议的通讯协议效率会更高。

实际上比较好的分布式框架，应该尽量适用公网和局域网通信的两种不同场景，Dubbo 就具备这种特征。



6. SOA 架构演化

1）单一应用架构

- 当网站流量很小时，只需一个应用，将所有功能都部署在一起，以减少部署节点和成本。
- 此时，用于简化增删改查工作量的 数据访问框架(ORM) 是关键。

2）垂直应用架构

- 当访问量逐渐增大，单一应用增加机器带来的加速度越来越小，将应用拆成互不相干的几个应用，以提升效率。
- 此时，用于加速前端页面开发的 Web 框架(MVC) 是关键。

3）分布式服务架构

- 当垂直应用越来越多，应用之间交互不可避免，将核心业务抽取出来，作为独立的服务，逐渐形成稳定的服务中心，使前端应用能更快速的响应多变的市场需求。

- 此时，用于提高业务复用及整合的 分布式服务框架(RPC) 是关键。

4）流动计算架构

- 当服务越来越多，容量的评估，小服务资源的浪费等问题逐渐显现，此时需增加一个调度中心基于访问压力实时管理集群容量，提高集群利用率。
- 此时，用于提高机器利用率的 资源调度和治理中心(SOA) 是关键。

7. Dubbo 优缺点

1）透明化的远程方法调用

- 像调用本地方法一样调用远程方法；只需简单配置，没有 API 侵入。

2）软负载均衡及容错机制

- 可在内网替代 nginx lvs 等硬件负载均衡器。

3）服务注册中心自动注册 & 配置管理

- 不需要写死服务提供者地址，基于接口名自动查询提供者。使用类似 zookeeper 等分布式协调服务作为服务注册中心，可以将绝大部分项目配置移入 zookeeper 集群。

4）服务接口监控与治理

- Dubbo-admin 与 Dubbo-monitor 提供了完善的服务接口管理与监控功能，针对不同应用的不同接口，可以进行 多版本，多协议，多注册中心管理。

8. Dubbo&HSF&Spring Cloud 对比

大家在做技术选型的时候，如果你选 Spring Cloud 做微服架构是可以的，它的整个体系比较完善，组件体系应该有 50 个左右非常多，而且比较完善，阿里也贡献了 Spring Cloud 一系列框架，整个社区项目非常完善，是做微服务的一个选择。

Dubbo 现在也开始支持 REST API，成为微服务有效版图中的一块。Dubbo 更强的是，做企业局域网分布式架构的大规模集群改造，用 Dubbo 协议层次上会做的更优秀。

淘宝的 HSF 相对来说生态没有那么好，主要是在阿里内部有，而 Dubbo 在社区内遍地开花，成为顶级项目，这是 Dubbo 比较优秀的地方。

	HSF	Dubbo	Spring Cloud
注册中心	Nacos	Eureka/Nacos/Zookeeper/Consul	Eureka/Nacos/Zookeeper/Consul
调用方式	RPC	RPC/SOAP/Rest	REST API
服务网关	Taobao Nginx	Nginx	Zuul/Gateway
断路器	Sentinel	Sentinel	Hystrix
分布式配置	diamond	Nacos	Spring Cloud Config
服务跟踪	淘宝鹰眼	zipkin	Sleuth/zipkin
消息总线		无	Spring Cloud Bus
数据流		无	Spring Cloud Stream
批量任务	Elastic-Job	Elastic-Job	Spring Cloud Task
部署宿主	Jboss定制化容器	多种部署方式	多种部署方式

2. Dubbo 分布式加与淘宝双 11 大规模服务集群治理

内容简介：

一、Dubbo 分布式架构

二、Dubbo 分布式集群架构

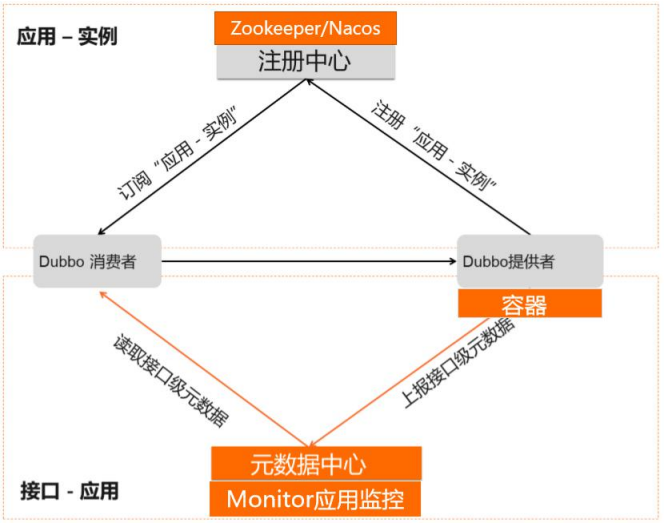
三、Dubbo 核心组件

四、Dubbo SOA 服务治理

五、Dubbo 未来底层同步与异步架构

六、Dubbo 分布式架构与 Remoting

一、Dubbo 分布式架构



Dubbo 架构主要分为两个部分：客户端与服务端，也可称为 Dubbo 消费者和 Dubbo 提供者。

例如电商网站，要调支付服务、物流服务、订单服务、监控服务、大数据服务等，服务是某些功能的封装，这些是提供者所提供的服务。作为客户端的话，用户要调用这些服务去完成某些业务功能，因为客户端和服务端是位于不同进程，位于分布式网络中不同的节点上，这是典型的一个架构。如果客户端和服务端两个都是单体的话，则为最简单的分布式架构。

随着业务的不断发展，架构也会不断演化。有些业务是大规模的集群，此时单点调用无法解决问题，这是由于服务端可能有 10 台、100 台、1000 台甚至更多。同样的服务有时候上线有时候下线，无法预估有多少台可能会上线，多少台会下线，此时需要使用集群的弹性收缩功能解决这个问题。

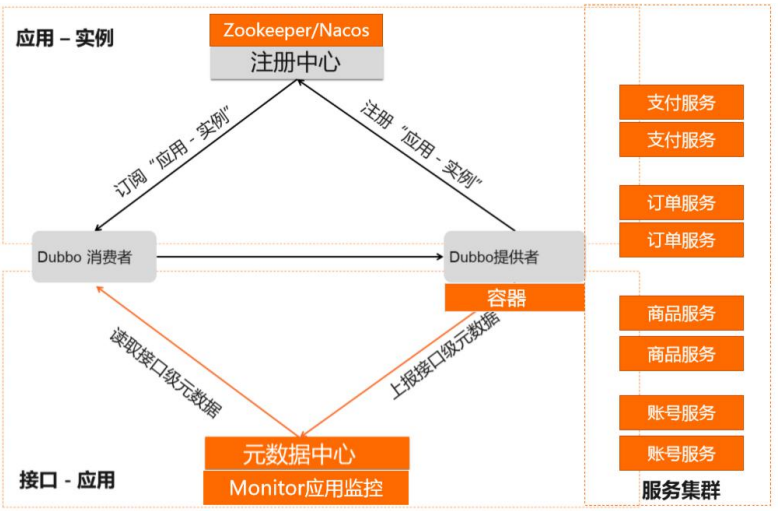
如果集群从上线到下线到运行一直都是 10 台，则正常情况就能满足需求。但如果在某些大型业务场景，例如双 11 时，平时 10 台可以满足的业务，在双 11 的时候可能需要 1000 台，甚至上万台进行支撑。因此，不同的业务集群规模是不一样的，小规模集群可能直接基于 IP 进行集群搭建就可以，大规模集群则需要对接多个不同的局域网，基于域名的方式进行调度的情况可能更多一点。

因此，Dubbo 分布式架构不仅仅是客户端和服务端、消费者和提供者这样的调用关系，它解决的是大规模客户端与大规模服务端的管理问题。

简单的架构要解决技术问题的话，它还需要有一个注册中心，还有应用监控中心，这就是典型的 Dubbo 早期解决问题。

二、Dubbo 分布式集群架构

在这个基础上，如果服务是不定数量的大规模集群，就需架构进一步去升级改造。



注册中心可能是个集群，监控中心可能是个集群，在大型业务场景下，服务可能也是个集群。

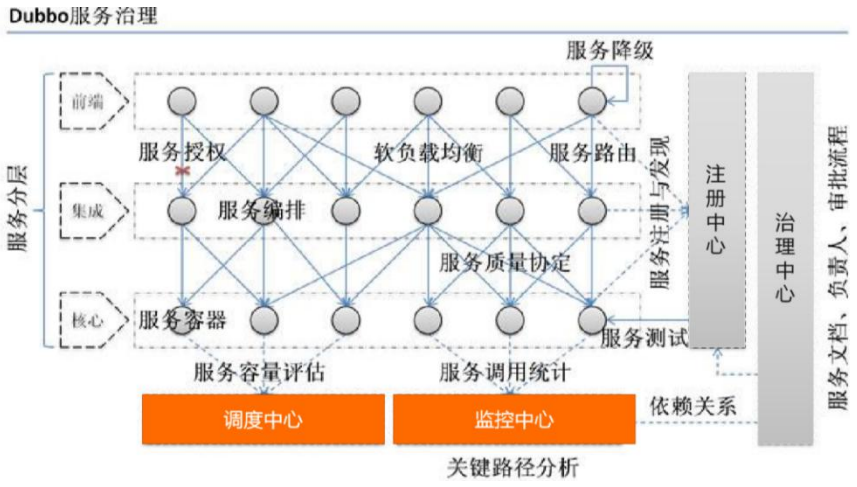
无论是支付服务、订单服务、商品服务、账号服务等，都可以根据自己的实际的并发需求，去部署不定规格、不定数量的大规模服务集群。Dubbo 联合其他技术人员给用户提供快速上线、包括管理的服务。Dubbo 不仅解决分布式调用的问题，还解决了多协议大规模集群的治理问题，这也是 Dubbo 在其他大型互联网公司中收到广泛好评的原因。

三、Dubbo 核心组件

Dubbo 主要有以下核心组件：

- Provider：服务的提供方，通过 Jar 或者容器的方式启动服务
- Consumer：服务消费方
- Registry：注册中心
- Monitor：统计服务和调用次数，调用时间监控中心。（Dubbo 的控制台页面中可以显示，目前只有一个简单版本）
- Container：服务运行的容器

四、Dubbo SOA 服务治理



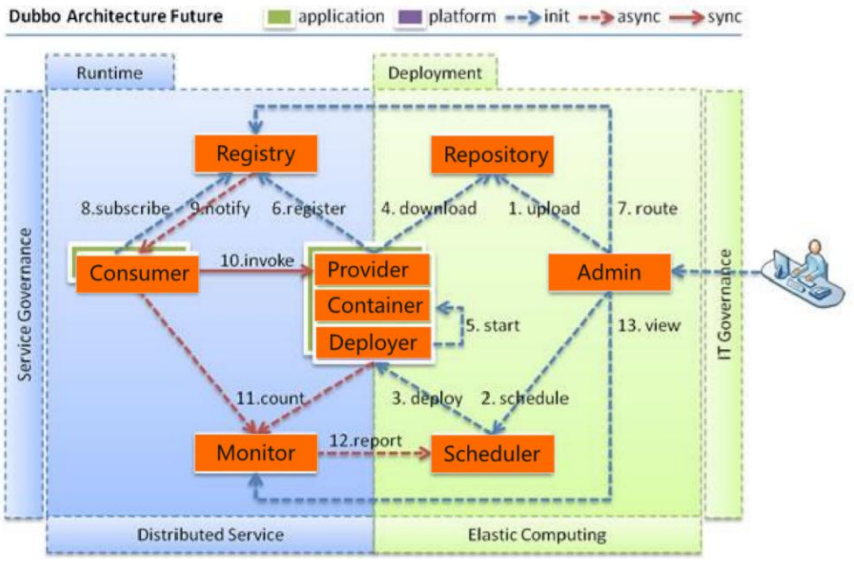
Dubbo 诞生于 SOA 盛行的 2008 年，早期基于 Web Service 中心 SOA 服务治理，后面在这个基础上逐渐开始大规模的扩充，服务之间关系越来越复杂。此时需要治理中

心，用来做大规模的流量调度、监控等，这些都是极具挑战性的问题，因此阿里开发了很多工具在内部去解决这些问题。

对于这些挑战性问题，Dubbo 的许多观点思想与目前微服务架构 Spring Cloud 不谋而合。这是由于当微服务到了大规模集群的层次上，无论怎么拆，它终究会面临这些问题，例如熔断限流，负载均衡，流量调度，安全治理等。

2013～2017 年 Dubbo 疏于维护，而这是 Spring Cloud 蓬勃发展时期，逐渐成为世界级微服务框架，但从某种程度上来说，Dubbo 可以称为 Spring Cloud 的前辈。

五、Dubbo 未来底层同步与异步架构



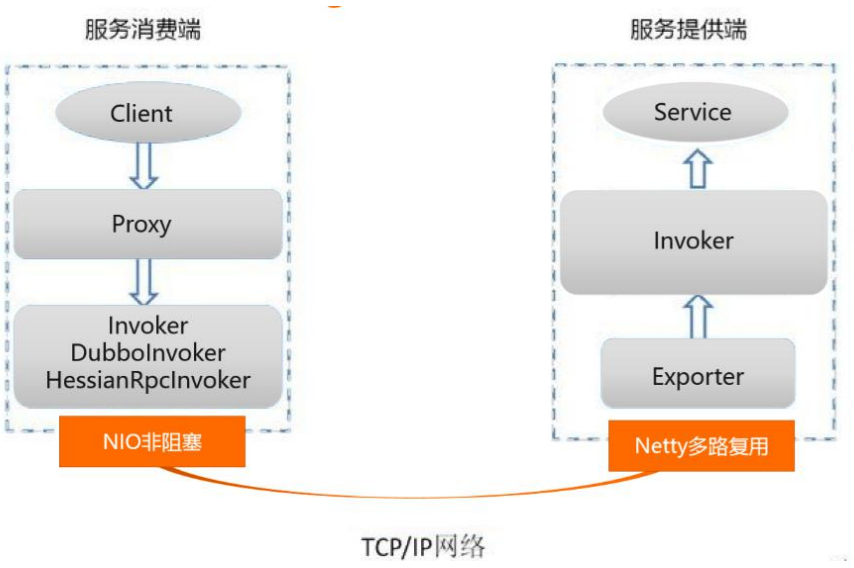
Dubbo 从 3.0 版本开始，也在做服务高并发、协议等方面的改造。如云原生支持的改造，去支持底层的高并发高吞吐量，去对接 K8s 云原生的工具，协议层支持 RPC

等新的类型，从而让 Dubbo 整个通信协议更多样化，使用户在不同场景下可以选择不同的实践方式。

异步请求和同步请求各有优势，异步请求优势在于非阻塞，同步请求的优势在于即时性，但如果是阻塞操作、长时间操作的话，可能会导致线程卡死的状况，影响整体的并发。

Dubbo 整个分布架构是更高层级的架构，它是大规模服务集群，不仅仅是开发落地，而且要大规模服务治理，它的实践经验给其他互联网公司的架构提供了很好的参考。

六、Dubbo 分布式架构与 Remoting



Dubbo 分布式架构的远程通讯（Remoting）提供对多种 NIO 框架抽象封装，包括“同步转异步”和“请求-响应”模式的信息交换方式。

3. Dubbo3.0 分布式 RPC 协议解析

内容简介：

一、Dubbo 分布式 RPC 协议解析

二、高级面试题：Dubbo 支持的协议

三、Dubbo 默认协议

四、Dubbo 线程消息分发模型

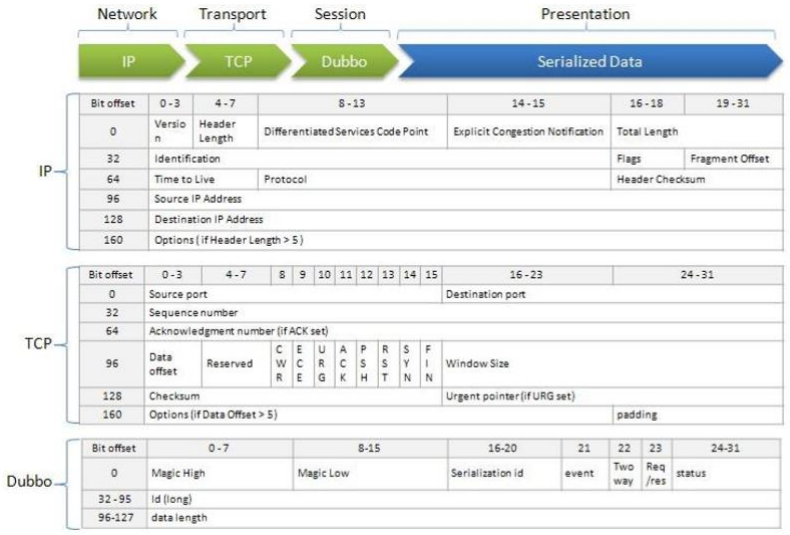
五、Dubbo 提供的注册中心

一、Dubbo 分布式 RPC 协议解析

Dubbo 是 RPC 基础上改造的一套框架，底层分时通信的基础设施没变，主要是在协议层另做了一层封装，封装思路是大规模服务集群治理，希望支持更多协议，不仅仅是 Webservice，或者 REST API 这种风格的通信，而是直接封装的服务接口，发送授信协议进行通信，基于汽车网络协议中的 MQTT 再次封装。

企业局域网通信，即使有公网通信，有些平台也希望定制自己的协议，典型代表有 QQ、微信和阿里钉钉，封闭的聊天系统里面通信很多，Dubbo 是自定义协议，实际在此基础上提供协议的再次封装，如在 TCP 基础上，不仅支持 REST，传统跨平台跨语言的企业级行业协议，另外还会做一些协议扩展，效率方面有更高要求，希望在原生的 TCP 或者 UDP 基础上，在二进制数据包上做封装。

Dubbo 的封装规定严格，华为自定了一套通信加密协议，协议基于 GDP 或 UDP 协议进行封装，是一种格式协议，只有自己能解析，QQ 或者微信的协议，只有腾讯能解析。Dubbo 思想也参考了经典分布式网络通信中的分包定义格式和思想规范。



如上图所示：

1-4 Version；

5-8 Header Length；

8-13 DifferentiatedServicesCodePoint；

14-15 ExplicitCongestionNotification；

16-18 ,19-31 Totall Length

...

每个比特位都有严格定义，这种严格的消息定义主要用于自定义， Dubbo 的通信

协议编码指的是 Dubbo 的自定义消息编码的格式规范。

同样解码需要获取 IP 地址、有效的消息载体，都有严格定义，定义的字段越丰富，消息通信的语义功能越强大，消息的封装也越复杂，如只留 1 个字节，8 个比特作为这个消息的额外信息，后面的话全是消息有效体，消息有效载体利用率就较高。

上图中，总共有 128 个比特，这里作为前置所有消息的扩展定义，第一阶段入 IP 包，然后 TCP 包的封装，Dubbo 消息包包括封装、序列化 ID、魔术编码的的比特位、状态码等。

Dubbo 在原始的 ip、tcp 协议之上进行了再次封装，后面在对象或者数据传时候做了序列化，序列成二进制格式，这样会效率更高，如果 JSON 就会涉及到更复杂的编码问题。实际上 Dubbo 的整个原生协议的封装足够复杂，128 个比特位自己定义用于 16 个字节，有效载荷就比较浪费。

Dubbo 除了自定义协议的设计思想外，也和其他几个典型聊天软件已对比，有的用标准协议，有的自定义协议，自定义协议封闭性更强效率更高，行业标准协议考虑跨平台的通用性及安全性问题。

二、高级面试题：Dubbo 支持的协议

Dubbo 支持多种协议：

1）基于 TCP 协议 Netty、Mina 实现

2）Dubbo 协议（默认）

3）Hessian 协议

4）HTTP 协议

5）RMI 协议

6）WebService 协议

7）Thrift 协议

8）Memcached 协议

9）Redis 协议

10）gPRC

11）Http2.0

Dubbo 协议支持足够丰富，现在做分布式开发，Dubbo 是非常优秀的分布式框架扩展，通信协议的支持已经在 GRPC 做的足够优秀，强大的地方在于做了大规模集群的服务治理， Dubbo 不仅是一套思想，更是能落地且经过阿里大规模服务实践的落地检验。

三、Dubbo 默认协议

1）Dubbo 协议默认使用 Hessian2 序列化。（说明： Hessian2 是阿里在 Hessian 基础上进行的二次开发， 起名为 Hessian2 ）

2）rmi 协议 默认为 java 原生序列

3）http 协议 默认为 json

4）hessian 协议，默认是 hessian 序列化

5）webservice 协议，默认是 soap 文本序列化

在局域网方面，Dubbo 的原则协议已经足够优秀，多种协议的好处在于可灵活根据自己的场景定制选择，允许自定义，为更复杂高级的场景允许扩展的接口。

四、Dubbo 线程消息分发模型

- Dispatcher 分发器
- all, direct, message, execution, connection
- ThreadPool 线程池
- fixed, cached

Dubbo 线程消息分发模型就是客户端和和服务端进行通讯，另外里面涉及到消息体的问题，消息体占多大，作为消息封装框架要考虑功能及效率问题。

五、Dubbo 提供的注册中心

- Multicast 注册中心
- Zookeeper 注册中心
- Redis 注册中心
- Simple 注册中心

4. Dubbo 分布式 Order 订单服务集群治理实战

内容简介：

一、Dubbo 分布式集群架构

二、Dubbo 分布式 Order Service 集群架构

三、Dubbo 订单服务集群调用实战

四、Dubbo 订单服务集群

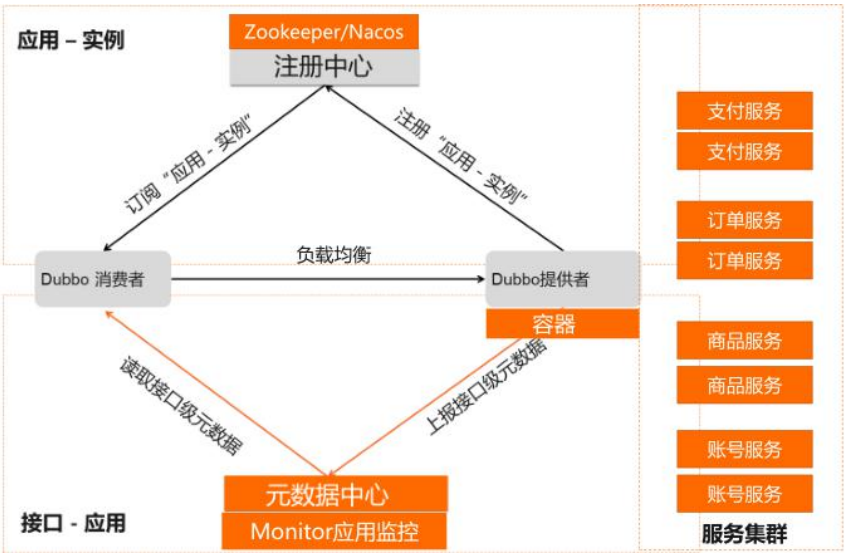
这节课模拟淘宝订单服务看如何使用 Dubbo，之前提到 Dubbo 有许多不同协议，包括核心部分注册中心、客户端、服务端。注册中心要先上线，类似微服务架构，可以用 zookeeper 注册服务中心注册。

一、Dubbo 分布式集群架构

实战开发首先安装 JAVA 开发环境，Dubbo 需要加部分依赖，因为 Dubbo 没有快速创建的整套模板，需要自己构建服务端、注册中心等，客户端需要添加依赖进行开发。

Dubbo 是典型的分布式集群架构，首先理清楚整个项目的结构关系，消费者是订单的调度端，服务端是订单的增、删、改、查的服务，注册中心可以采用 Zookeeper 也可采用 Nacos，具体项目可以继续扩展，如加入订单服务、支付服务、商品服务、账号服

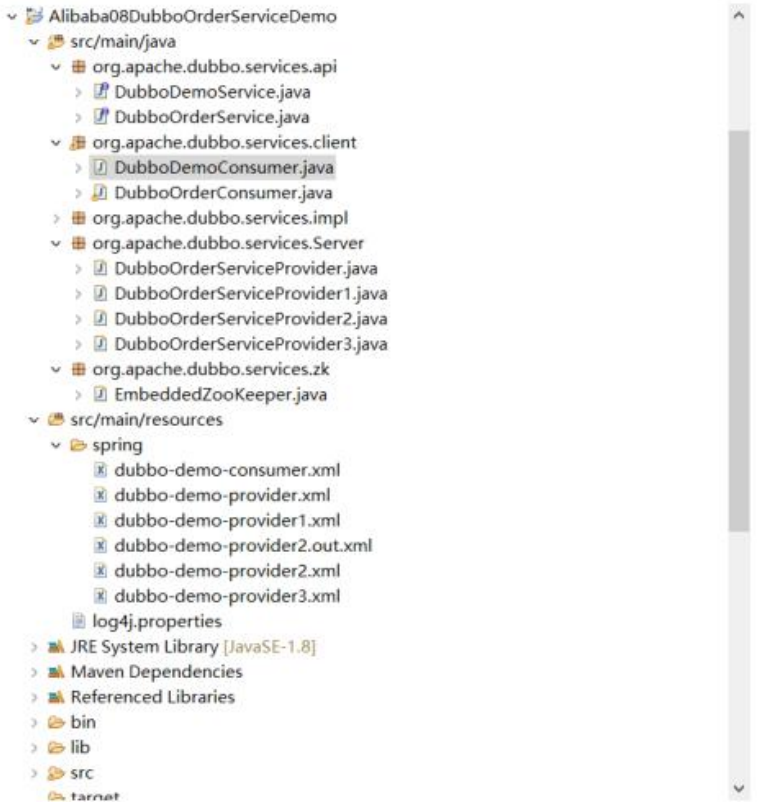
务等相关服务集群。目前先模拟一个服务，然后再到多个服务，单条线路跑通再做周边的服务开发。



二、Dubbo 分布式 Order Service 集群架构

Dubbo 模拟订单服务，有服务的订单就是服务端，需要 host 容器，可直接用自定义开发的控制台程序，需要在服务端定服务实现或服务接口，客户端要实现调度端或有代理对象。注意，还需要 Zookeeper 注册中心，可用内嵌的 Zookeeper 服务，也可用独立部署的 Zookeeper 服务。

服务端的配置文件和客户端的配置文件要配置，服务端配置文件主要是：服务的端口、地址以及需要用哪些服务类型。客户端配置文件主要是：要调用的服务，注册中心的位置使用什么协议等关键参数。



三、Dubbo 订单服务集群调用实战

模拟整个开发工作，API 端是服务接口，客户端是客户端程序，客户端程序通用 API 代理，后面会调用后端真正的服务，真正的服务托管在 Provider，是远程服务提供方，规范服务类型托管、运行、接受请求。

如下图所示，Provider 复制了 4 份，参考其中一个代码，是 main 函数。

```java
package org.apache.dubbo.services.Server;

import org.apache.dubbo.services.zk.EmbeddedZooKeeper;
import org.springframework.context.support.ClassPathXmlApplicationContext;
/**
 * @Title Alibaba Dubbo Service Interface服务接口
 * @Author Frank 侠客
 * @Desc 阿里巴巴Dubbo大规模集群服务治理
 */
public class DubboOrderServiceProvider {

    public static void main(String[] args) throws Exception {
        //1.启动内嵌模拟ZK
        new EmbeddedZooKeeper(2181, false).start();
        System.out.println("Dubbo集群Service0 Host启动了");
        @SuppressWarnings("resource")
        ClassPathXmlApplicationContext context = new ClassPathXmlApplicationContext(new String[]{"spr
        context.start();
        System.in.read(); // press any key to exit
    }
}
```

启动后有自己的主进程去加载 Bean，启动接受请求，会模拟起用 Zookeeper，Zookeeper 是嵌入式模拟注册中心。配置文件对应的是 provider，看下核心参数：

```xml
    http://dubbo.apache.org/schema/dubbo http://dubbo.apache.org/schema/dubbo/dubbo.xsd">

<!-- provider's application name, used for tracing dependency relationship -->
<dubbo:application name="pay-provider">
    <dubbo:parameter key="qos.enable" value="true" />
    <dubbo:parameter key="qos.accept.foreign.ip" value="false" />
    <dubbo:parameter key="qos.port" value="22220" />
</dubbo:application>

<!-- use multicast registry center to export service -->
<dubbo:registry group="aaa"
    address="zookeeper://127.0.0.1:2181" />
<dubbo:registry address="zookeeper://127.0.0.1:2181" />
<!--<dubbo:registry address="zookeeper://11.163.250.27:2181"/> -->

<!-- use dubbo protocol to export service on port 20880 -->
<dubbo:protocol name="dubbo" port="20890" />

<!-- service implementation, as same as regular local bean -->
<bean id="payService" class="org.apache.dubbo.services.impl.DubboPayServiceImpl"/>
<!-- service implementation, as same as regular local bean -->
<bean id="orderService" class="org.apache.dubbo.services.impl.DubboOrderServiceImpl"/>

<!-- declare the service interface to be exported -->
<dubbo:service interface="org.apache.dubbo.services.api.DubboPayService" ref="payService"/>
<dubbo:service interface="org.apache.dubbo.services.api.DubboOrderService" ref="orderService"/>
```

需要配置一下 Zookeeper 注册中心地址，使用的协议是 Dubbo 原生协议。然后配置服务，服务接口暴露给客户端使用。

之前复制了 4 份 provider，后面多启动几个服务，主要用于模拟集群，比如订单服务要起用 3 台、30 台、300 台，是一对多的过程。同理都自己的配置文件，因为在同一台机器上，配置不一样，端口要变， 在同一台机器上模拟各个不同端口。如下图所示，端口是“20893”：

```
    </dubbo:application>

    <!-- use multicast registry center to export service -->
    <dubbo:registry group="aaa"
        address="zookeeper://127.0.0.1:2181" />
    <dubbo:registry address="zookeeper://127.0.0.1:2181" />
    <!--<dubbo:registry address="zookeeper://11.163.250.27:2181"/> -->
    <!-- use dubbo protocol to export service on port 20880 -->

    <!-- 提供方应用信息，用于计算依赖关系 -->
    <dubbo:application name="dubbo-order-service-demo"  />
    <dubbo:registry address="nacos://${nacos.address:127.0.0.1}:8848"/>
    <dubbo:config-center address="nacos://${nacos.address:127.0.0.1}:8848"/>
    <!-- 如果要使用自己创建的命名空间可以使用下面配置 -->
    <!-- <dubbo:registry address="nacos://10.20.153.10:8848?namespace=5cbb70a5-xxx-xxx-xxx-d43479ae0932"

    <dubbo:protocol name="dubbo" port="20890" />
    <!-- service implementation, as same as regular local bean -->
    <bean id="payService" class="org.apache.dubbo.services.impl.DubboPayServiceImpl"/>
  <!-- service implementation, as same as regular local bean -->
    <bean id="orderService" class="org.apache.dubbo.services.impl.DubboOrderServiceImpl"/>

    <!-- declare the service interface to be exported -->
    <dubbo:service interface="org.apache.dubbo.services.api.DubboPayService" ref="payService"/>
    <dubbo:service interface="org.apache.dubbo.services.api.DubboOrderService" ref="orderService"/>


</beans>
```

模拟几台机器一个集群，当这几台实例上线以后，都会 Zookeeper 进行注册，输入成功以后客户端进行调用。客户端调用订单服务，订单通过 ID 查询订单打印下返回字符串。具体项目可以通过 JDBC 等链接数据库。

调度端里面是 Dubbo 的订单接口，模拟死循环，不断循环向客户端发请求。

```
public static void main(String[] args) {
    @SuppressWarnings("resource")
    ClassPathXmlApplicationContext context = new ClassPathXmlApplicationContext(new
    context.start();
    DubboDemoService demoService = (DubboDemoService) context.getBean("demoService")
    System.out.println("Dubbo集群Client启动了");
    int i =0;
    while (true) {
        i++;
        try {
            Thread.sleep(1000);
            String result = demoService.sayHello("Dubbo"); // call remote method
            System.out.println(result);
            Thread.sleep(1000);
            String order = demoService.getOrderById("taobao-2046-000"+i); // call re
            System.out.println(order);

        } catch (Throwable throwable) {
            throwable.printStackTrace();
```

客户端代理对象是 proxy，proxy 起客户端代理作用，获取配置文件 Beanba 的配置信息，调取方法：通过创建对象调取订单接口，根据 ID 模拟调用情况返回结果。循环调用实际上只起用了一个客户端，只不过每隔一秒调用一次。

```
import org.apache.dubbo.services.api.DubboOrderService;
import org.springframework.context.support.ClassPathXmlApplicationContext;
/**
 * @Title Alibaba Dubbo Service Interface服务接口
 * @Author 侠客
 * @Desc 阿里巴巴Dubbo大规模集群服务治理
 */
public class DubboOrderConsumer {

    public static void main(String[] args) {
        @SuppressWarnings("resource")
        ClassPathXmlApplicationContext context = new ClassPathXmlApplicationContext(new String[]{"spri
        context.start();
        DubboOrderService orderService = (DubboOrderService) context.getBean("orderService"); // get r
        System.out.println("Dubbo订单OrderClient启动了");
        while (true) {
            try {
                Thread.sleep(1000);
                String order = orderService.getOrderById("taobao-2046-0001"); // call remote method
                System.out.println(order);

            } catch (Throwable throwable) {
                throwable.printStackTrace();
            }
        }
    }
}
```

四、Dubbo 订单服务集群

模拟集群，调用服务端一台机器的效果，调用客户端，虽经过注册中心，但客户端经过注册中心调用后端。如下图所示，每隔一秒调后端，显示“20890”：

```
DubboDemoConsumer [Java Application] C:\Program Files\Java\jre1.8.0_131\bin\javaw.exe  (2021-4-30 20:41:40)
Dubbo集群Client启动了
Hello Dubbo, Dubbo集群Service服务地址: 10.6.102.177:20892
订单查询Id：taobao-2046-0001, Dubbo集群OrderService服务地址: 10.6.102.177:20892
Hello Dubbo, Dubbo集群Service服务地址: 10.6.102.177:20891
订单查询Id：taobao-2046-0002, Dubbo集群OrderService服务地址: 10.6.102.177:20892
Hello Dubbo, Dubbo集群Service服务地址: 10.6.102.177:20892
订单查询Id：taobao-2046-0003, Dubbo集群OrderService服务地址: 10.6.102.177:20892
Hello Dubbo, Dubbo集群Service服务地址: 10.6.102.177:20890
订单查询Id：taobao-2046-0004, Dubbo集群OrderService服务地址: 10.6.102.177:20892
Hello Dubbo, Dubbo集群Service服务地址: 10.6.102.177:20892
订单查询Id：taobao-2046-0005, Dubbo集群OrderService服务地址: 10.6.102.177:20890
Hello Dubbo, Dubbo集群Service服务地址: 10.6.102.177:20892
订单查询Id：taobao-2046-0006, Dubbo集群OrderService服务地址: 10.6.102.177:20891
Hello Dubbo, Dubbo集群Service服务地址: 10.6.102.177:20890
```

说明这台服务器只有一台，再怎么轮询负载均衡都在一台机器。这时启动第二台服务器，逐步上线更多服务，如下图所示，有“20890”、“20891”，两台机器都被调用了。

再上线第三台，客户端本身加载最新服务列表有时间，这里会有延迟，涉及服务上线和客户端发现最新的服务列表的过程，这个机制跟微服务架构很像。

这里面第二台服务器已经出来，正常生产环境下，客户端和服务端通过注册中心解耦，可进行服务集群的灵活扩容，平时只有 100 台，但在双 11 的时候可以加到 1000 台。体现了 Duddo 相比传统简单架构，往更高级别灵活性弹性集群架构进行升级扩展，Duddo 还有性能监控功能、生产环境的上线下线等功能。在当年背景下，阿里能够做出这种框架，并且在生产环境下大规模验证（双 11 验证）非常牛。

分析整个 Duddo 架构，整个设计思想解决的问题，比当前的微服务架构协议更灵活、部署方式更灵活，架构更原生。现在 Spring Cloud 在全球范围内使用更广、生态文档更完善，但实际上 Duddo 体系更单一，功能更丰富，性能更高。Duddo 的并发性一定比 Spring Cloud 高，Duddo 可以灵活的在局域网和公网之间协议切换。

现在 Duddo 结合阿里其他框架，逐步做分布式大规模集群治理生态的完善工作，虽然有些已经开始对接 Spring Cloud 微服务，但并不是重点。严格来说，Spring Cloud 体系里面的协议，只是 Duddo 的一种，Duddo 后面会做的越来越好 Duddo3.0 版本在做的云原生、服务治理、安全与性能监控等模式支持都非常优秀。

5. Dubbo 集成 Nacos 注册中心实战

内容简介：

一、Dubbo 集成 Nacos 注册中心

二、Dubbo 集成 Nacos 注册中心实战

这节课讲的是 Dubbo 集成 Nacos 注册中心实战，Nacos 当前在微服务领域非常有名，不仅仅支持 Spring Cloud，而且支持 Dubbo 服务集群，当然也支持 GO 语言等客户端进行集成，是非常优秀的注册中心架构。另外 Nacos 还可以做配置服务，对标携程开源的阿波罗注册中心，大规模集服务集群配置向外迁移的时候可以使用 Spring Cloud、Nacos 等服务。

一、Dubbo 集成 Nacos 注册中心

1. Nacos 注册中心

Nacos 是阿里巴巴开源的新一代分布式服务注册和查找架构，提供配置和管理微服务，构建以“服务”为中心的现代应用架构等等，功能非常强大，总结如下：

1）Nacos 微服务动态服务发现、配置管理和服务管理工具平台。

2）Nacos 致力于帮助发现、配置和管理微服务。

3）Nacos 帮助更敏捷和容易地构建、交付和管理微服务平台。Nacos 是构建以“服务”为中心的现代应用架构（例如微服务范式、云原生范式）的服务基础设施。

4）Nacos 支持几乎所有主流的“服务”的发现、配置和管理。

5）Kubernetes Service。

6）gRPC & Dubbo RPC Service。

7）Spring Cloud RESTful Service。

8）https://nacos.io。

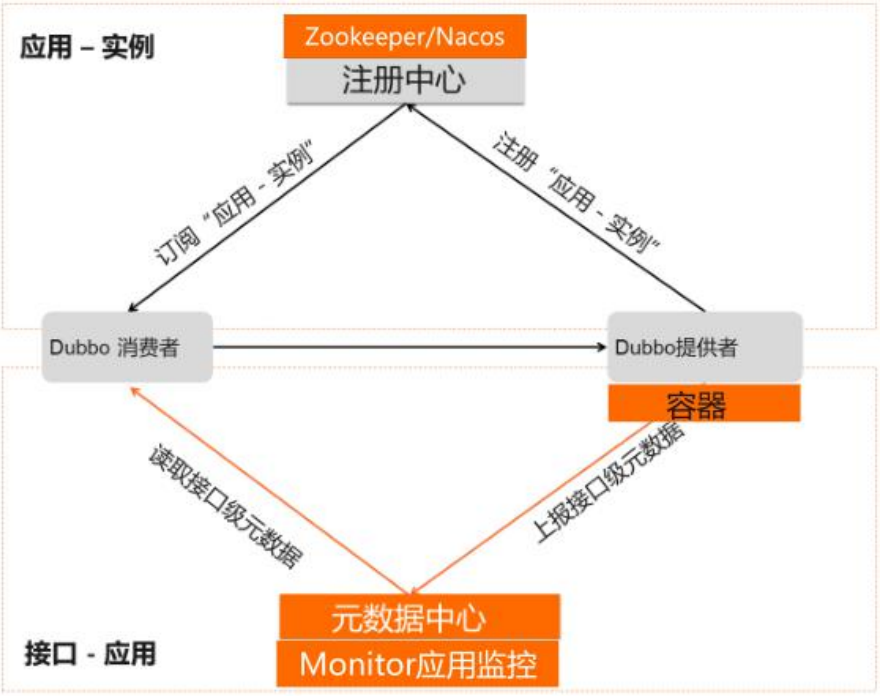
2. Dubbo 集成 Nacos 注册中心

Dubbo 和 Nacos 进行集成，很重要的一点是，Dubbo 作为分布式服务治理框架，可以写服务接口，Dubbo 本身与 Nacos 并没有交集，Nacos 起到注册中心和配置中心的作用，但是有统一标准的通信协议。提交数据做集成时，Nacos 专门为 Dubbo 开发 Nacos 客户端，集成时依赖 Nacos 客户端，然后配置一下，向注册中心提交数据。Nacos 作为 Dubbo 生态系统中重要的注册中心实现，其中 dubbo-registry-nacos 则是 Dubbo 融合 Nacos 注册中心的实现。

3. Dubbo 分布式架构

回顾 Dubbo 分布式架构架构，最简单的架构就是调用端、服务端、注册中心，向外扩展有很多功能，如注册中心，应用监控、大规模服务集群的负载均衡等。这是 Dubbo

比较牛逼的地方。因为在 Spring Cloud 大规模流行之前，实际 Dubbo 已经实现解决了这些问题，Dubbo 本身也是一个生态，很多功能并没有在自己的范围内解决。阿里开源其他框架，帮助协调解决大规模服务集群的治理工作。



二、Dubbo 集成 Nacos 注册中心实战

1. Dubbo 集成 Nacos 实战步骤

Dubbo 集成 Nacos 实战的关键步骤，首页构造一个 Dubbo 服务，给 Dubbo 配置 Nacos 客户端依赖，再配置 Nacos 注册中心的地址，还需要提前启动 Nacos 注册中心

的服务，然后 Dubbo 服务才可以在 Nacos 注册中心里面进行注册。推荐使用 Dubbo 2.6.5 以上的版本，不建议再用之前的版本，涉及 Nacos 客户端依赖，一般使用 1.2.0 以上版本，最新是 1.4.0，太旧的版本不推荐用，容易遇到兼容性等问题。步骤总结如下：

- 启动 Nacos 注册中心；
- Dubbo 改造加入 dubbo-registry-nacos 依赖；
- dubbo-registry-nacos 的 Maven 依赖 pom.xml 文件中；
- 推荐您使用 Dubbo 2.6.5+。

Dubbo 的 pom.xml 文件：

```
<dependencies>

    ...

    <!-- Dubbo Nacos registry dependency -->
    <dependency>
        <groupId>com.alibaba</groupId>
        <artifactId>dubbo-registry-nacos</artifactId>
        <version>0.0.2</version>
    </dependency>

    <!-- Keep latest Nacos client version -->
    <dependency>
        <groupId>com.alibaba.nacos</groupId>
        <artifactId>nacos-client</artifactId>
        <version>[0.6.1,)</version>
    </dependency>

    <!-- Dubbo dependency -->
    <dependency>
        <groupId>com.alibaba</groupId>
        <artifactId>dubbo</artifactId>
        <version>2.7.8</version>
    </dependency>

    <!-- Alibaba Spring Context extension -->
    <dependency>
        <groupId>com.alibaba.spring</groupId>
        <artifactId>spring-context-support</artifactId>
        <version>1.4.0</version>
    </dependency>

    ...

</dependencies>
```

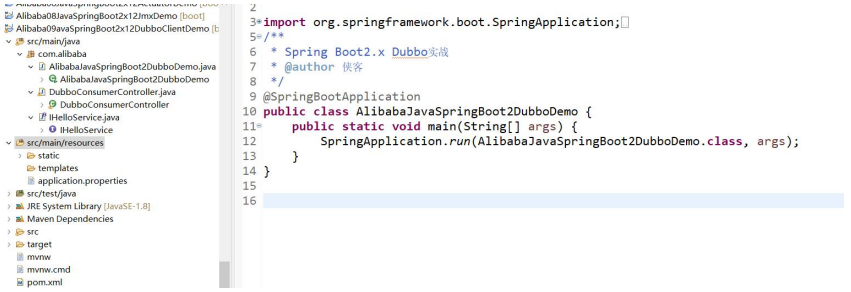
2. Nacos 监控 Dubbo 服务

上线完成以后，Nacos 自带 web 的界面，可以打开看一下整个服务列表，还有配置管理等一系列复杂的功能。启动服务注册里面有详情的详情、删除等操作功能，可以进行更详细的管理工作。

3. 实例演示：

官方参考资料：http://dubbo.apache.org/。官方的例子比较复杂，需要很多依赖包，这里演示一个简单的例子。

首先构建一个 Dubbo 服务，Dubbo 服务没有分包，直接放在根目录下面，有接口实现，放在同一个目录下面，就是一个“sayHello”包。传一个字符串，就返回一个字符串，基本上有一个实现类型。



这里面基于是“SringBoot”，做了一个 Dubbo 的服务端，Dubbo3.0 之后在进化，支持 Spring Cloud 微服务架构体系，可以提供 REST API，也可以做云原生，比较灵活。

注意看一下配置文件，两个关键点，一个是 Dubbo 的起步依赖，选的是 2.7.8，一个是 Nacos 选的是 1.2.1。

```
        </dependencies>
    </dependencyManagement>

    <dependencies>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-web</artifactId>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-actuator</artifactId>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-devtools</artifactId>
        </dependency>
        <dependency>
            <groupId>org.apache.dubbo</groupId>
            <artifactId>dubbo-spring-boot-starter</artifactId>
            <version>2.7.8</version>
        </dependency>
        <dependency>
            <groupId>com.alibaba.nacos</groupId>
            <artifactId>nacos-client</artifactId>
            <version>1.2.1</version>
        </dependency>
    </dependencies>
```

配置文件，首先要配置自己的名字，这里是“dubbo-provider-demo”，还有指定扫描的包，这里是叫“com.alibaba”,下面是 nacos 的一个例“nacos://127.0.0.0:8848”。

```
spring.application.name=SpringBoot2xDubboDemo
server.port=9000
#management.endpoints.jmx.domain=com.alibaba
#management.endpoints.jmx.unique-names=true
# Base packages to scan Dubbo Components (e.g @Service , @Reference)
dubbo.scan.basePackages=com.alibaba
dubbo.application.name=dubbo-provider-demo
dubbo.registry.address=nacos://127.0.0.1:8848
```

这时需要提前启动 nacos，下载完成以后有一个解压包，可以直接启动。注意是单点模式启动，直接双击，有一个脚本，我们在 windows 上直接启动就 ok。看一下窗口界面有一个地址，这个地址是启动完成以后会有一个管理界面，第一次登录需要输入用户、密码。

把这个地址粘贴到网页，转到 Nacos 管理界面，现在看服务列表，里面还没有服务，因为服务还没有上线。

现在可以启动服务端，客户端到服务端需要连注册中心找到服务端进行调用。涉及包依赖，主要是如下两个依赖：

```
            <groupId>org.apache.dubbo</groupId>
            <artifactId>dubbo-spring-boot-starter</artifactId>
            <version>2.7.8</version>
        </dependency>
        <dependency>
            <groupId>com.alibaba.nacos</groupId>
            <artifactId>nacos-client</artifactId>
            <version>1.2.1</version>
        </dependency>
    </dependencies>
```

造成 REST API 作为调用的控制器出口，这里可以传一个字符串，然后让后台通过服务代理来调，里面的接口实现一模一样。如果在同一个工程里面，可以依赖同一个包，可以把 Controller 拆出来。这里基本没变，还是“SringBoot” 程序。

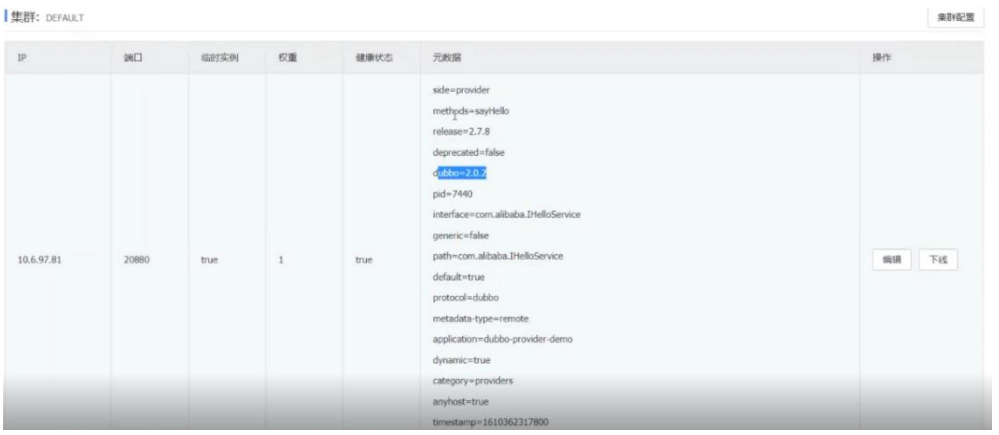
```
package com.alibaba;

import org.springframework.boot.SpringApplication;
/**
 * Spring Boot2.x Dubbo实战
 * @author 侠客
 */
@SpringBootApplication
public class AlibabaJavaSpringBoot2DubboDemo {
    public static void main(String[] args) {
        SpringApplication.run(AlibabaJavaSpringBoot2DubboDemo.class, args);
    }
}

```

下面是“Dubbo-client-demo”client 的意思是消费者，意思就是调用客户端的评论或支付信息。

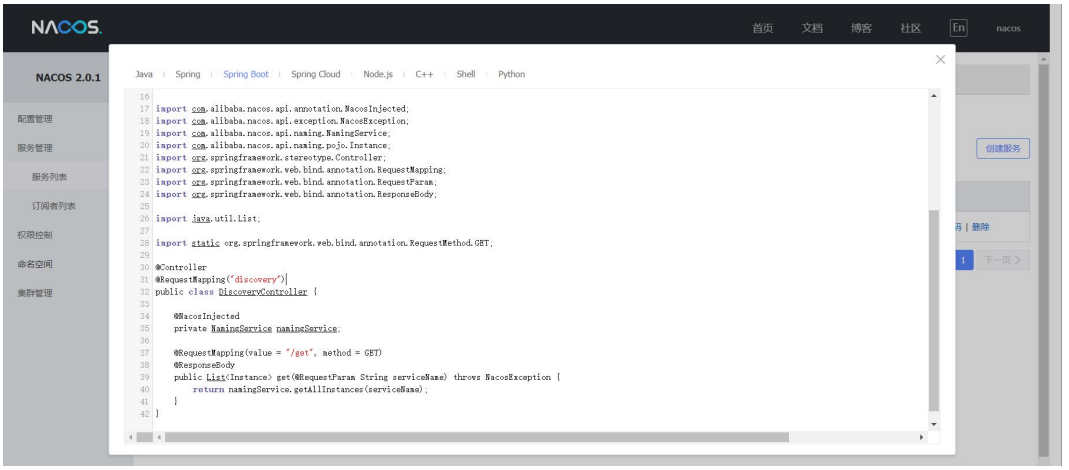
```
spring.application.name=SpringBoot2xDubboDemo
server.port=9001
#management.endpoints.jmx.domain=com.alibaba
#management.endpoints.jmx.unique-names=true
# Base packages to scan Dubbo Components (e.g @Service , @Reference)
dubbo.scan.basePackages=com.alibaba
dubbo.application.name=dubbo-client-demo
dubbo.registry.address=nacos://127.0.0.1:8848
```

这样一个过程，就是一次调用，也可以称为是一次命令“Order”，在电商系统里面“Order”被翻译成订单。模拟 Java 架构中，服务端向注册中心注册一个端口，注意端口不能重复，过程中不能出现错误。完成后刷新一下，Nacos 界面服务列表中已经显示出来我们的“com.alibaba”服务,说明服务端的服务已经上线。

点打看一下详情，可以看到用到的方法、release 等服务相关的描述信息，使用非常方便。



点击服务列表界面的“示例代码”，会告诉你如果通过其他服务做请求，代码应该怎么写，还有微服务架构怎么集成，这是 Nacos 非常优秀的地方。



接下来上线客户端，启动客户端来模拟服务调用，上传一个字符串，严格来说是一个代理。这里叫做“proxy”，实际是服务端的接口，客户端通过“proxy”接口生成一个变量创建实例，严格的说是创建一个代理对象，用于调取后台的远程服务接口。

```
@RestController
    public class DubboConsumerController {

        @Reference
        private IHelloService proxy;

        @RequestMapping("/sayHello/{name}")
        public String sayHello(@PathVariable String name) {
            return proxy.sayHello(name);
        }
    }
```

模拟启动客户端，传入字符串，调用接口，到 Nacos 服务界面刷新，可以看到服务列表里多出一条“consumers”服务，属于调用端。

端口是“9001”，地址是”sayHello”，然后通过 Dubbo 传送，返回结果正常，换成 Java，返回结果也是正常的。正常情况下，字符串是从服务端返回来的，客户端通过代理对象调用远程服务，获取信息。这是典型的 RPC 架构,用 Nacos 注册中心，非常容易就可以实现。

扫一扫

免费领取同步课程

钉钉扫一扫

进入官方答疑群

开发者学院【Alibaba Java 技术图谱】

更多好课免费学

阿里云开发者“藏经阁”

海量电子书免费下载